Symfony Messenger Workshop

Web Summer Camp 2019

Samuel Rozé

VP Engineering @ Birdie

Symfony
ContinuousPipe
ApiPlatform

Workshop's master plan

Workshop's master plan

1. Brief introduction to Symfony Messenger

Workshop's master plan

1. Brief introduction to Symfony Messenger

2. We code.

What's Symfony Messenger about?

What's Symfony Messenger about?

1. **Messages**. Any PHP object.

What's Symfony Messenger about?

1. **Messages**. Any PHP object.

2. **Message bus**. Where you dispatch your messages.

What's Symfony Messenger about?

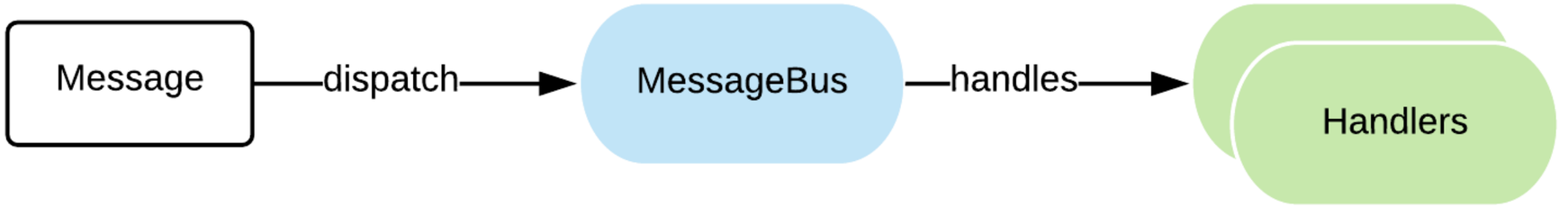
1. **Messages**. Any PHP object.
2. **Message bus**. Where you dispatch your messages.
3. **Message handlers**. Will execute your business logic when the message arrives to them.

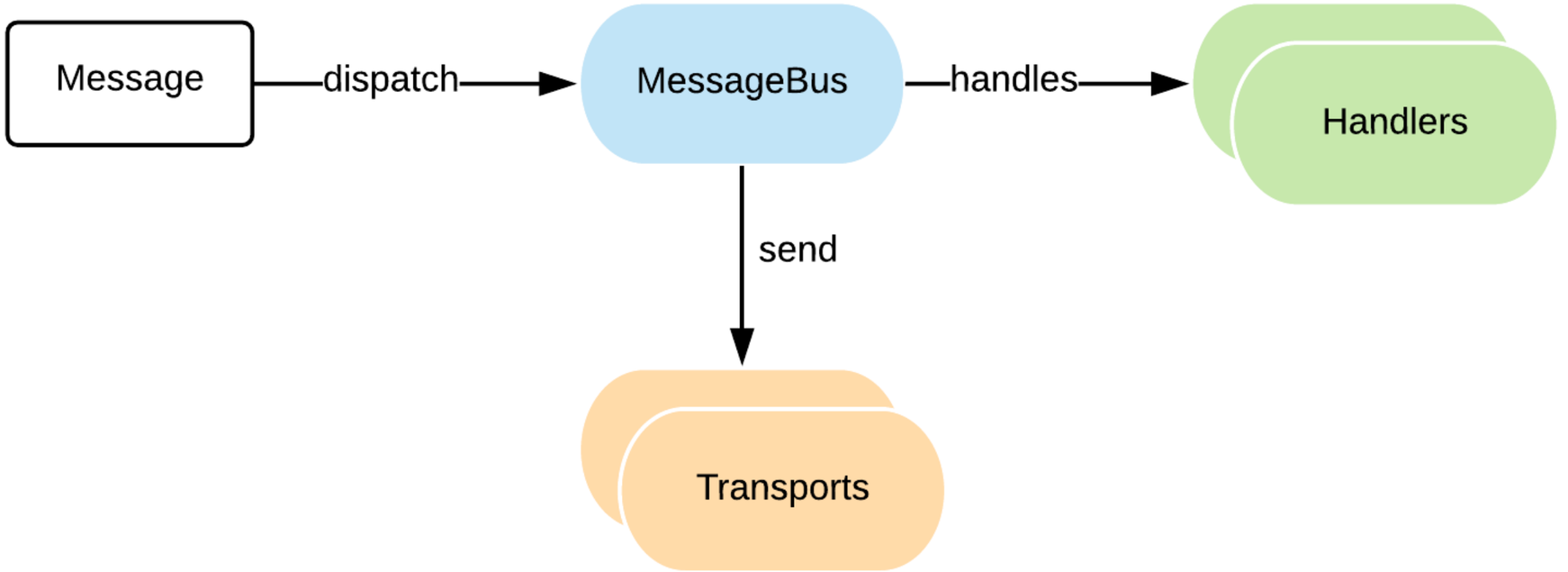
What's Symfony Messenger about?

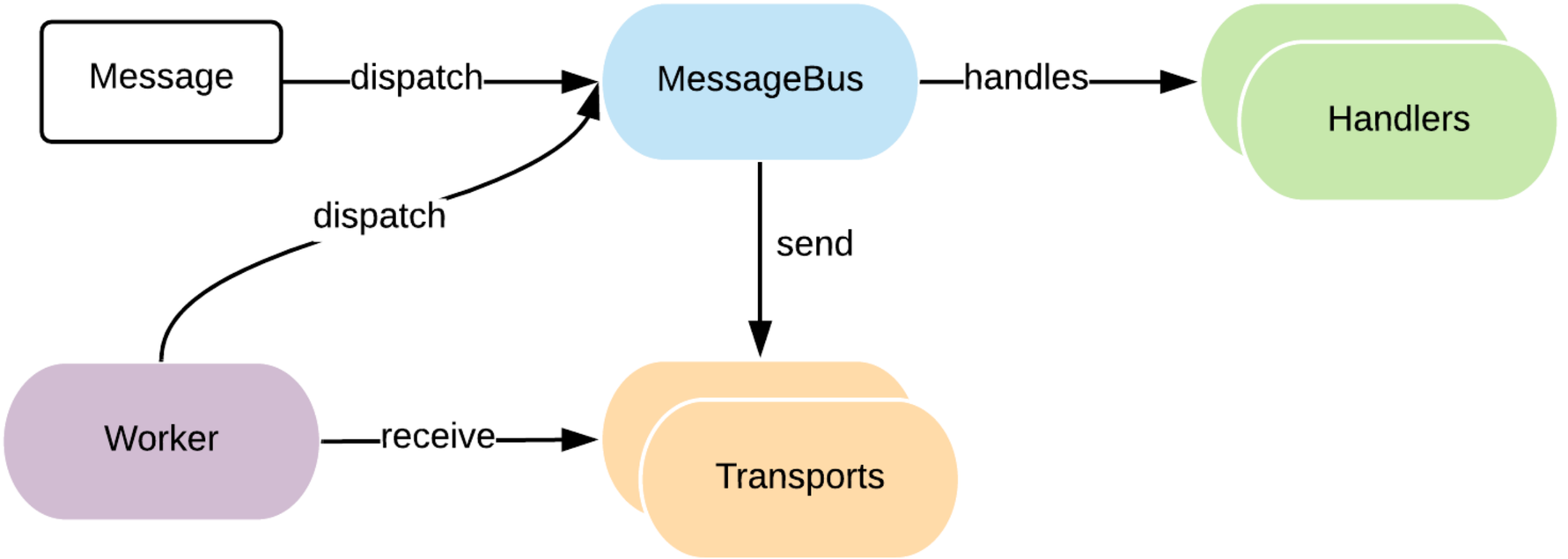
1. **Messages**. Any PHP object.
2. **Message bus**. Where you dispatch your messages.
3. **Message handlers**. Will execute your business logic when the message arrives to them.
4. **Transports**. Allow to send and receive messages through 3rd party systems.

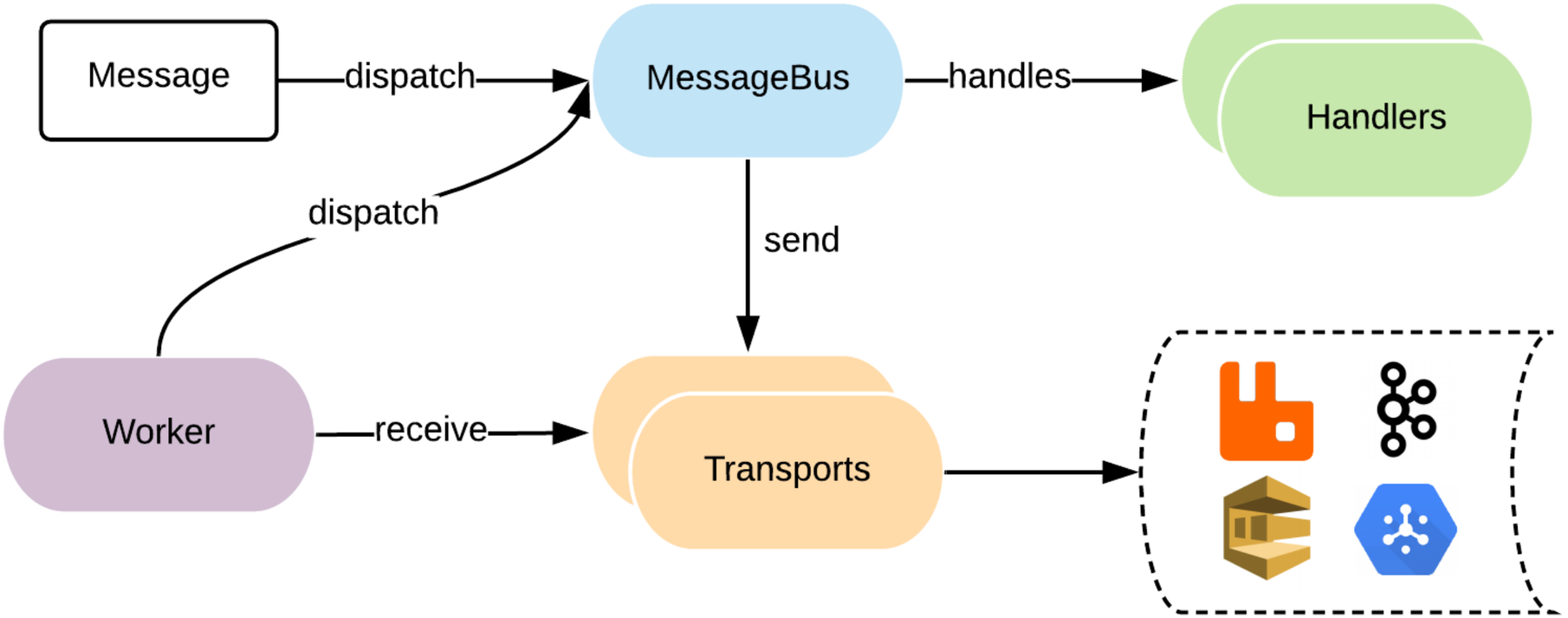
What's Symfony Messenger about?

1. **Messages**. Any PHP object.
2. **Message bus**. Where you dispatch your messages.
3. **Message handlers**. Will execute your business logic when the message arrives to them.
4. **Transports**. Allow to send and receive messages through 3rd party systems.
5. **Worker.** To consume messages from transports.


Message
dispatch
MessageBus
handles
Handlers


Message
dispatch
MessageBus
handles
Handlers
send
Transports


Message
dispatch
MessageBus
handles
Handlers
dispatch
send
Worker
receive
Transports


Message
dispatch
MessageBus
handles
Handlers
dispatch
send
Worker
receive
Transports


@samuelroze - #websc

A message

No specific requirement.

```php
namespace App\Message;

class SendNotification
{
    // ...properties

    public function __construct(string $message, array $users)
    {
        $this->message = $message;
        $this->users = $users;
    }

    // ...getters
}
```

As a component

```php
use Symfony\Component\Messenger\MessageBus;
use Symfony\Component\Messenger\HandlerLocator;
use Symfony\Component\Messenger\Middleware\HandleMessageMiddleware;

$handler = function(MyMessage $message) {
    // ...
};

$messageBus = new MessageBus([
    new HandleMessageMiddleware(new HandlerLocator([
        SendNotification::class => $handler,
    ])),
]);
```

In your Symfony application

Nothing to do...

Dispatching a message

```php
namespace App\Controller;

use Symfony\Component\Messenger\MessageBusInterface;
// ...

class DefaultController
{
    public function index(MessageBusInterface $bus, Request $request)
    {
        $bus->dispatch(new SendNotification(/* ... */));

        return new Response('<html><body>OK.</body></html>');
    }
}
```

Dispatching a message

```php
namespace App\Controller;

use Symfony\Component\Messenger\MessageBusInterface;
// ...

class DefaultController
{
    public function index(MessageBusInterface $bus, Request $request)
    {
        $bus->dispatch(new SendNotification(/* ... */));

        return new Response('<html><body>OK.</body></html>');
    }
}
```

Symfony Exception | Symfony Docs | Symfony Support

NoHandlerForMessageException | HTTP 500 Internal Server Error

No handler for message "App\Message\SendNotification".

Exception | Logs 1 | Stack Trace

Symfony\Component\Messenger\Exception\

NoHandlerForMessageException

in /Users/samuelroze/git/symfony/src/Symfony/Component/Messenger/**ContainerHandlerLocator.php** (line 36)

```
31.     {
32.         $messageClass = get_class($message);
33.         $handlerKey = 'handler.'.$messageClass;
34.
35.         if (!$this->container->has($handlerKey)) {
36.             throw new NoHandlerForMessageException(sprintf('No handler for message "%s".', $messageClass));
37.         }
38.
39.         return $this->container->get($handlerKey);
40.     }
41. }
```

ContainerHandlerLocator->**resolve**(*object*(SendNotification))
in /Users/samuelroze/git/symfony/src/Symfony/Component/Messenger/Middleware/**HandleMessageMiddleware.php** (line 34)

HandleMessageMiddleware->**handle**(*object*(SendNotification), *object*(Closure))
in /Users/samuelroze/git/symfony/src/Symfony/Component/Messenger/**MessageBus.php** (line 56)

A message handler

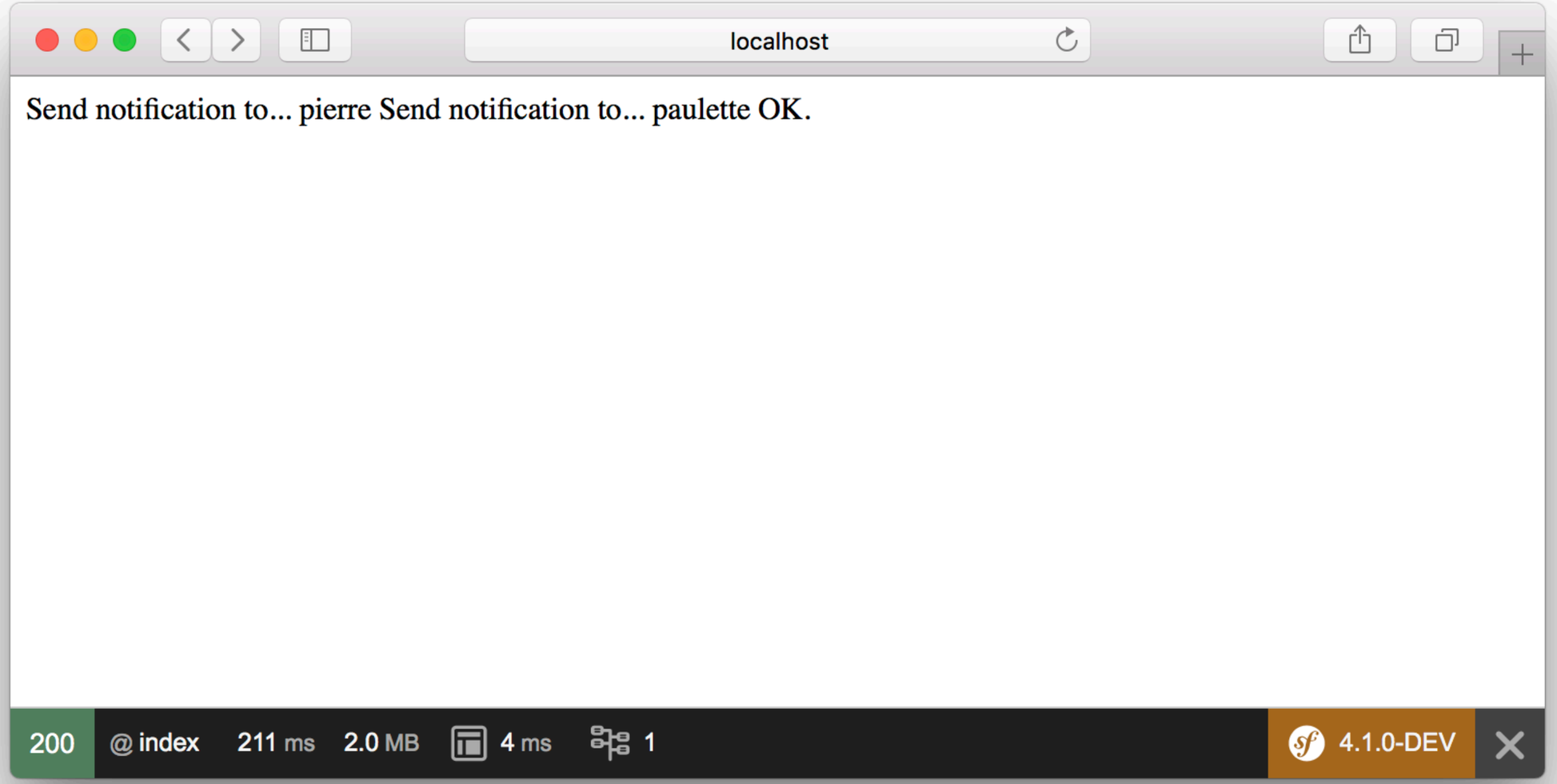
```php
namespace App\MessageHandler;

use App\Message\SendNotification;

class SendNotificationHandler
{
    public function __invoke(SendNotification $message)
    {
        foreach ($message->getUsers() as $user) {
            echo "Send notification to... ".$user."\n";
        }
    }
}
```

Register your handler

```yaml
# config/services.yaml
services:
    App\MessageHandler\SendNotificationHandler:
        tags:
            - messenger.message_handler
```

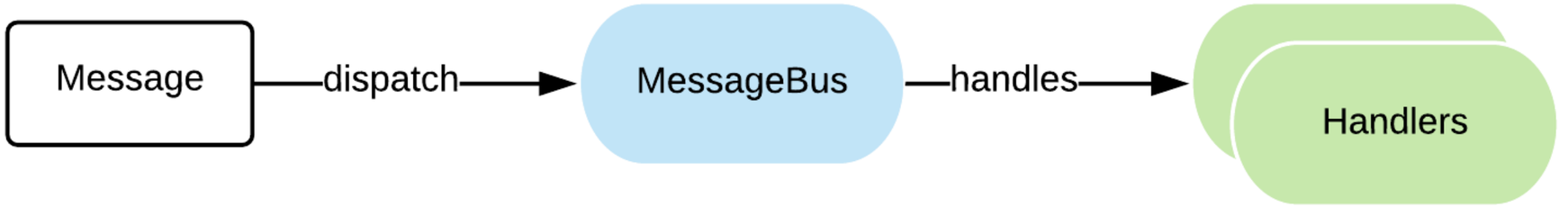
Register your handlers (option 2)

```yaml
# config/services.yaml
services:
    App\MessageHandler\:
        resource: '../src/MessageHandler/*'
        tags: ['messenger.message_handler']
```

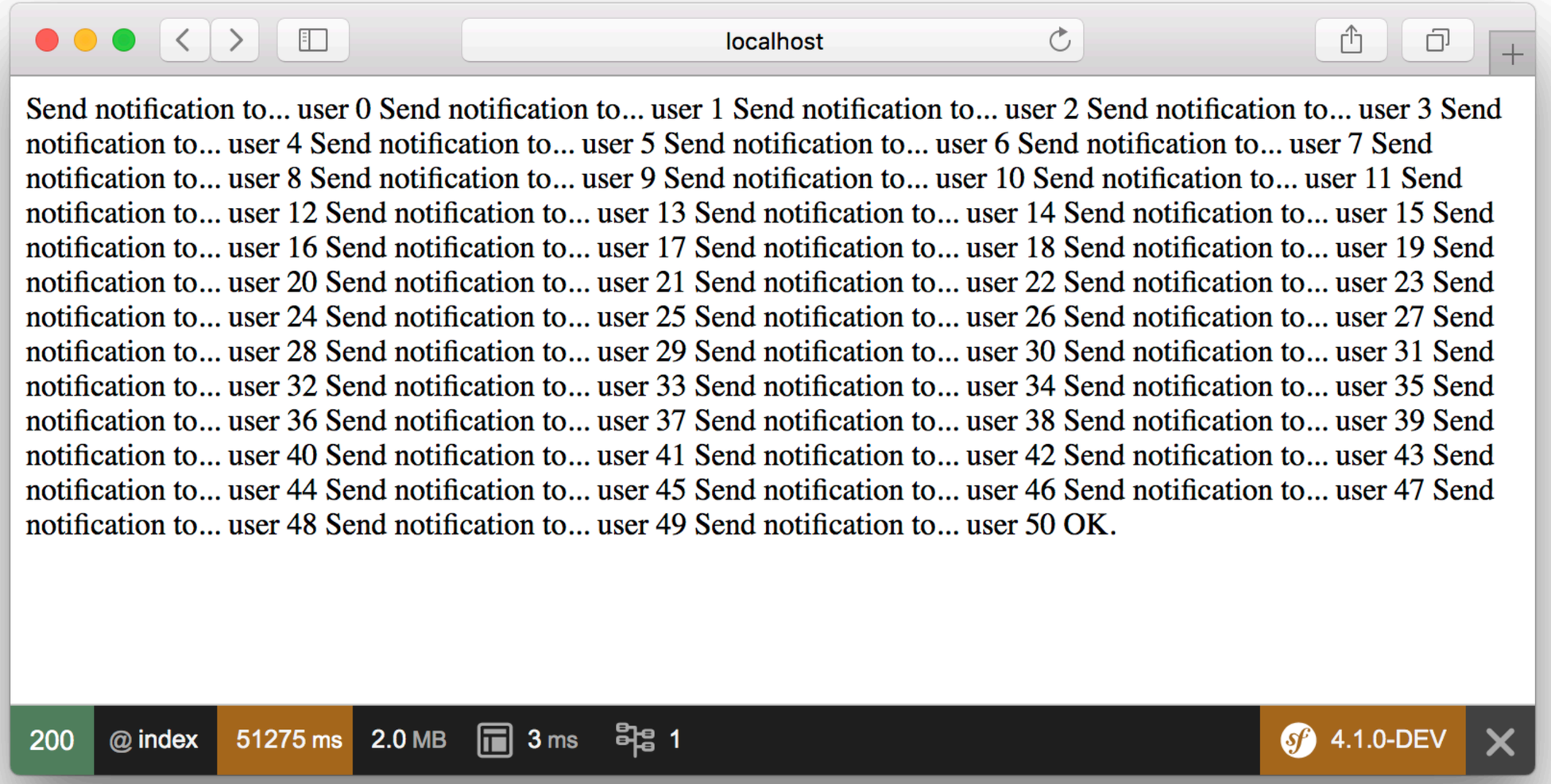

localhost
Send notification to... pierre Send notification to... paulette OK.
200
@ index
211 ms
2.0 MB
4 ms
1
4.1.0-DEV

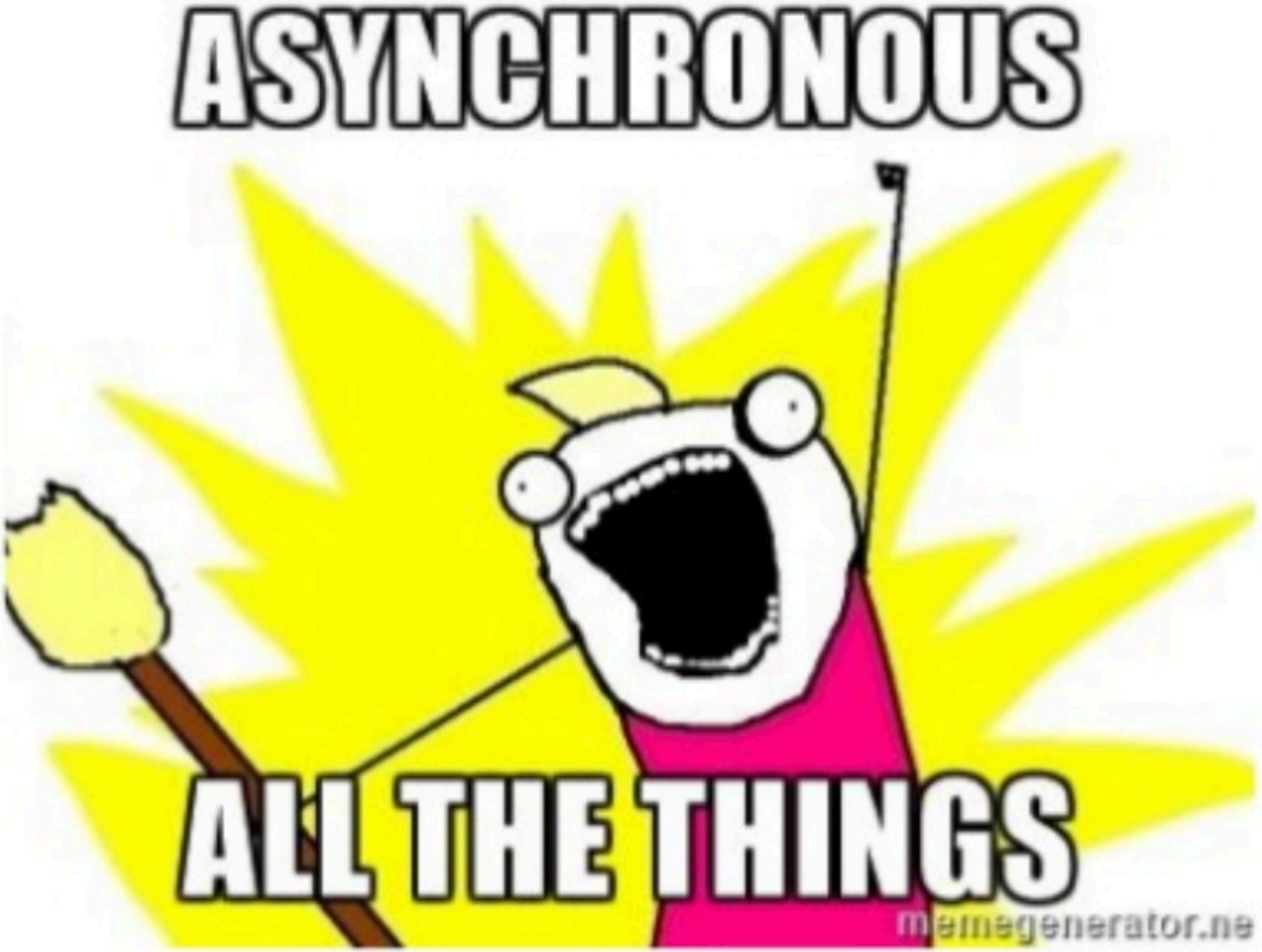
Yay, dispatched !

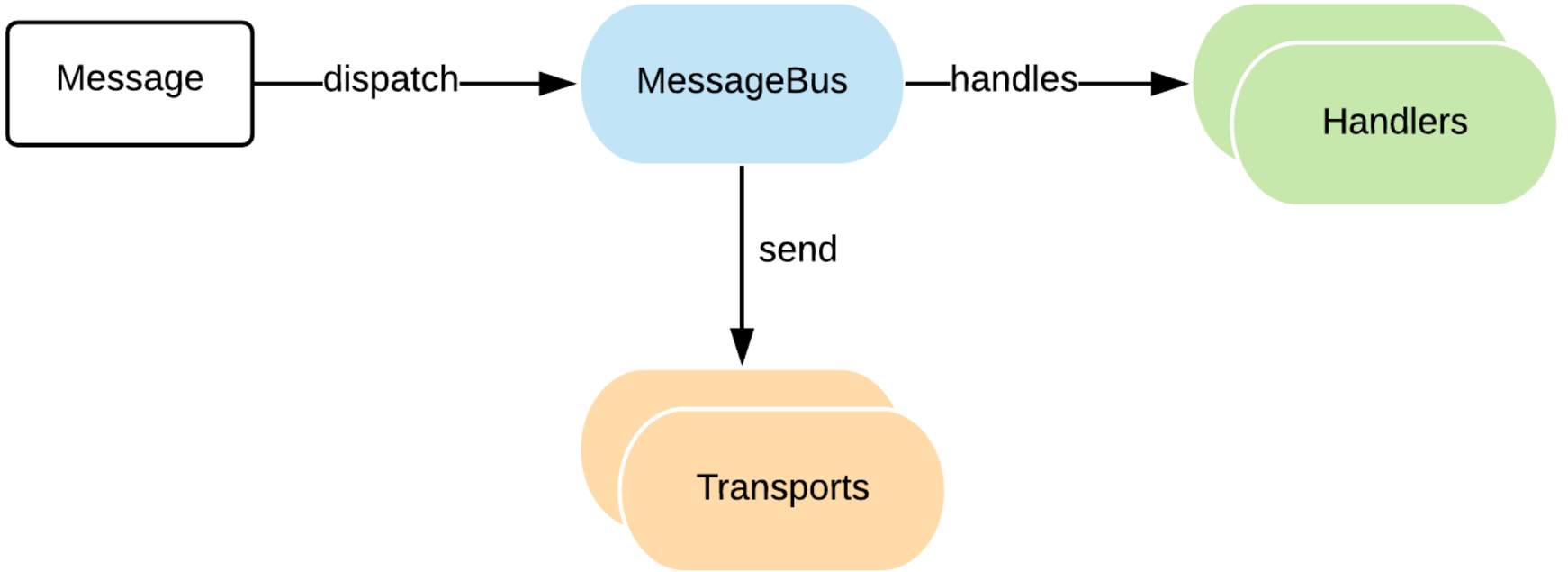
Handler is called. Done.


Message
dispatch
MessageBus
handles
Handlers

Now, imagine your handler takes **ages** to run.


localhost
Send notification to... user 0 Send notification to... user 1 Send notification to... user 2 Send notification to... user 3 Send notification to... user 4 Send notification to... user 5 Send notification to... user 6 Send notification to... user 7 Send notification to... user 8 Send notification to... user 9 Send notification to... user 10 Send notification to... user 11 Send notification to... user 12 Send notification to... user 13 Send notification to... user 14 Send notification to... user 15 Send notification to... user 16 Send notification to... user 17 Send notification to... user 18 Send notification to... user 19 Send notification to... user 20 Send notification to... user 21 Send notification to... user 22 Send notification to... user 23 Send notification to... user 24 Send notification to... user 25 Send notification to... user 26 Send notification to... user 27 Send notification to... user 28 Send notification to... user 29 Send notification to... user 30 Send notification to... user 31 Send notification to... user 32 Send notification to... user 33 Send notification to... user 34 Send notification to... user 35 Send notification to... user 36 Send notification to... user 37 Send notification to... user 38 Send notification to... user 39 Send notification to... user 40 Send notification to... user 41 Send notification to... user 42 Send notification to... user 43 Send notification to... user 44 Send notification to... user 45 Send notification to... user 46 Send notification to... user 47 Send notification to... user 48 Send notification to... user 49 Send notification to... user 50 OK.
200
@ index
51275 ms
2.0 MB
3 ms
1
4.1.0-DEV


ASYNCHRONOUS
ALL THE THINGS
memegenerator.ne


Message
dispatch
MessageBus
handles
Handlers
send
Transports

What's a transport?

What's a transport?

1. **Send & Receive**.
   Sends and receives… messages.

What's a transport?

1. **Send & Receive**.
   Sends and receives… messages.

2. **Is configurable via a DSN**
   So we have a unique and common way of configuring them.

What's a transport?

1. **Send & Receive**.
   Sends and receives… messages.

2. **Is configurable via a DSN**
   So we have a unique and common way of configuring them.

3. **Uses a serializer**
   By defaut, uses Symfony Serializer. Replace as you wish.

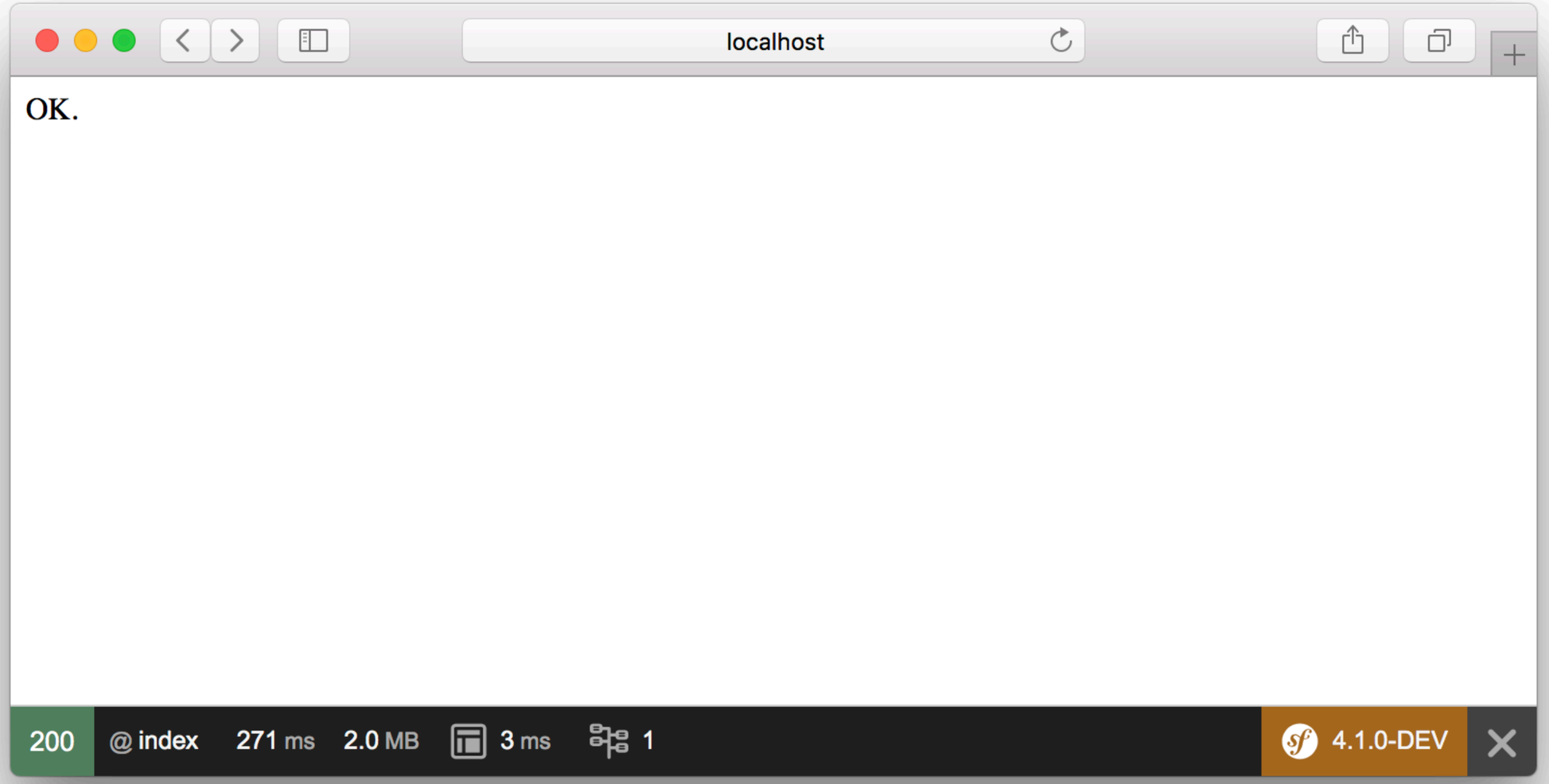
Configure your transport(s)

```yaml
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            myqueue: '%env(MESSENGER_DSN)%'
```

```
# .env
MESSENGER_DSN=amqp://guest:guest@localhost:5672/%2f/messages
```

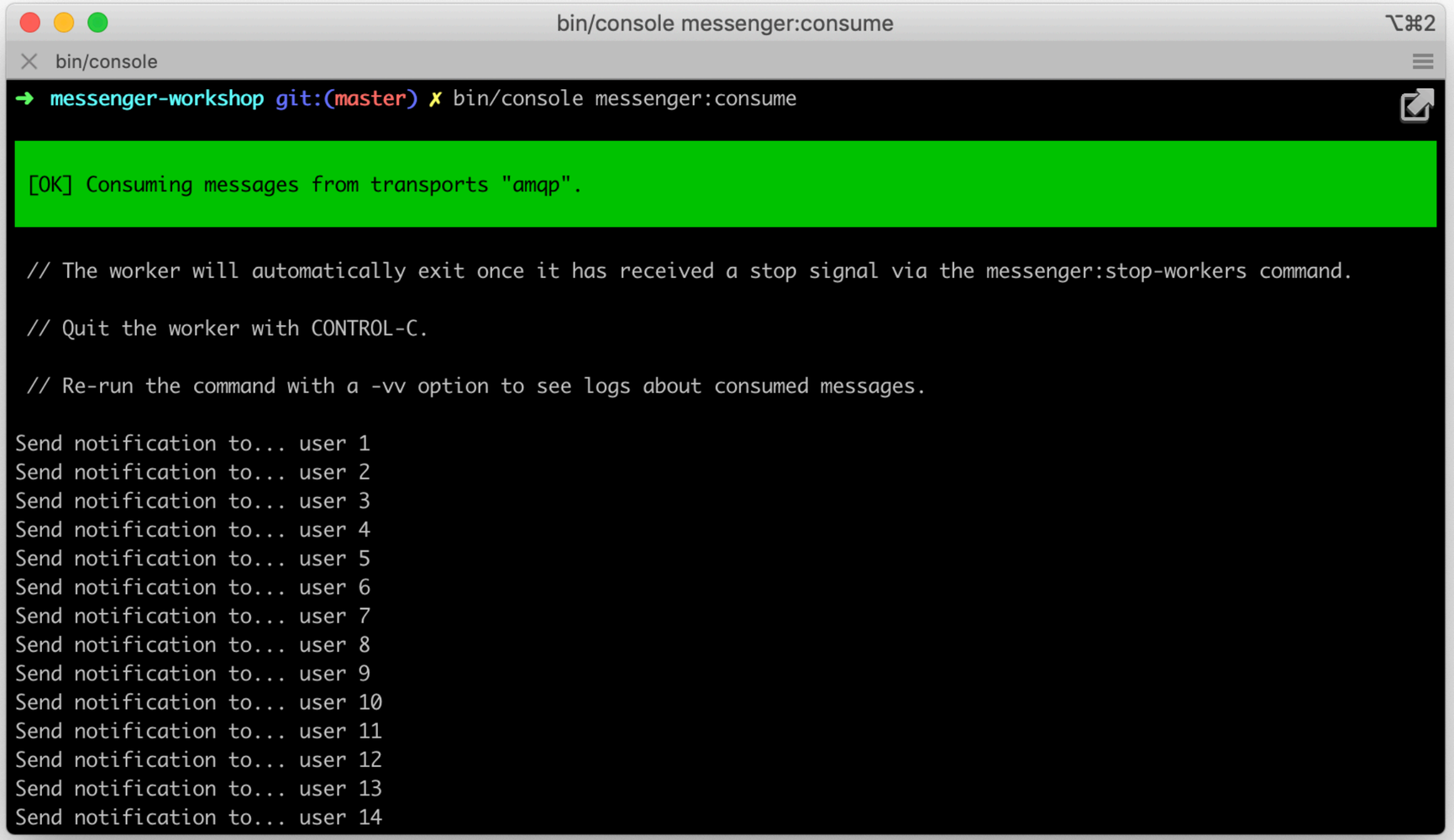
Route your message(s)

```
framework:
    messenger:
        routing:
            'App\Message\SendNotification': myqueue
```


localhost
OK.
200
@ index
271 ms
2.0 MB
3 ms
1
4.1.0-DEV


BUT... IT DIDN'T
DO ANYTHING, RIGHT?

```
$ bin/console messenger:consume
```


bin/console messenger:consume
⌥⌘2
bin/console
➜ messenger-workshop git:(master) ✗ bin/console messenger:consume
[OK] Consuming messages from transports "amqp".
// The worker will automatically exit once it has received a stop signal via the messenger:stop-workers command.
// Quit the worker with CONTROL-C.
// Re-run the command with a -vv option to see logs about consumed messages.
Send notification to... user 1
Send notification to... user 2
Send notification to... user 3
Send notification to... user 4
Send notification to... user 5
Send notification to... user 6
Send notification to... user 7
Send notification to... user 8
Send notification to... user 9
Send notification to... user 10
Send notification to... user 11
Send notification to... user 12
Send notification to... user 13
Send notification to... user 14


@samuelroze - #websc - 2019

Let's have the application running!

https://github.com/sroze/messenger-workshop

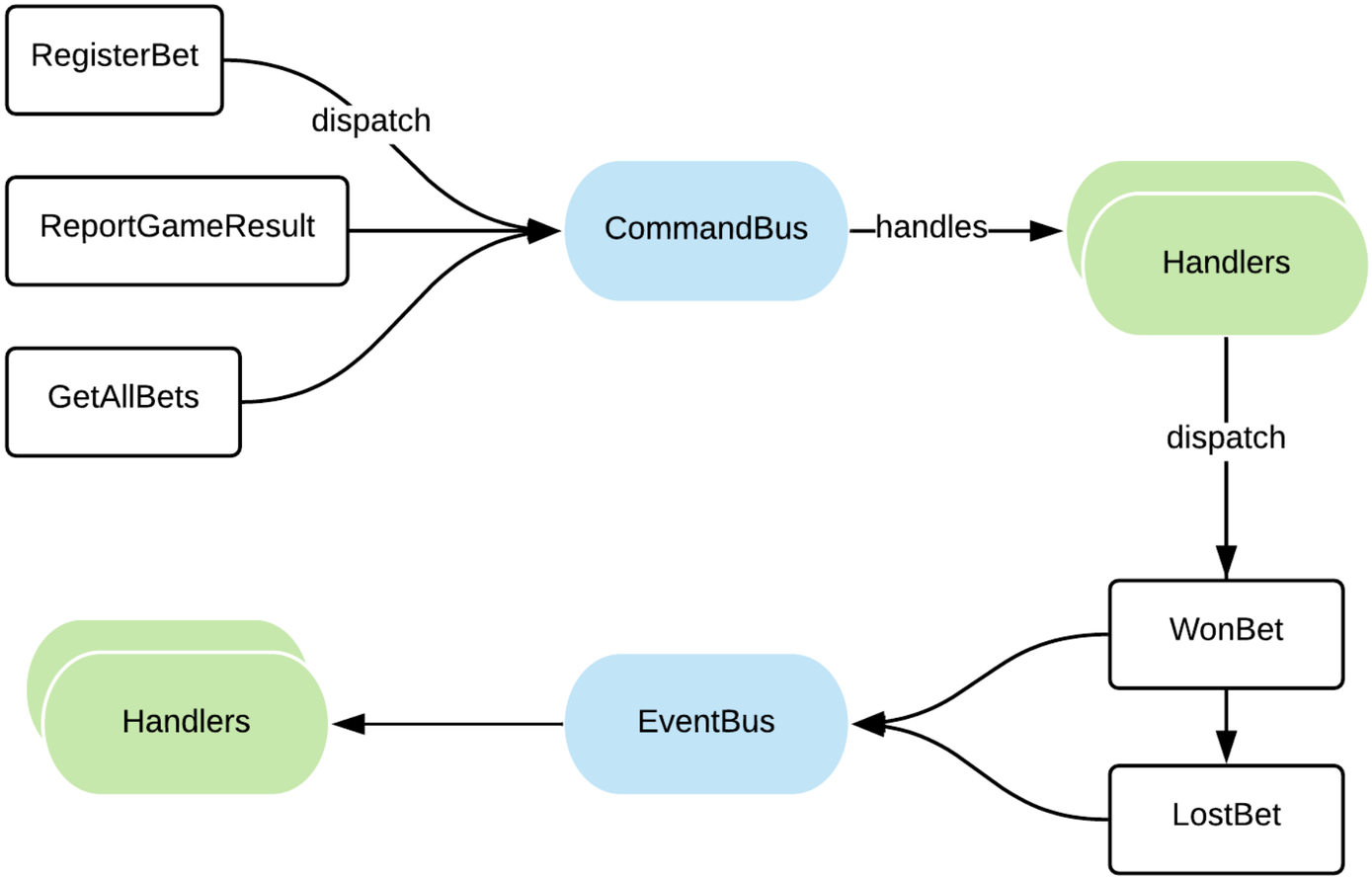
```
$ git pull origin master
```

```
$ bin/console server:run
```

Our application.

A very simple application that will be able to track bets on football games. Via the user interface, we will:

1. Register bets on different games
2. Submit the actual result of a game
3. Send a notification to the winners and losers


RegisterBet
dispatch
ReportGameResult
CommandBus
handles
Handlers
GetAllBets
dispatch
WonBet
Handlers
EventBus
LostBet

Points of attention.

Points of attention.

1. `$handledStamp = $envelope->last(HandledStamp::class);`

Points of attention.

1. `$handledStamp = $envelope->last(HandledStamp::class);`
2. `RegisterBetHandler` just implements `MessageHandlerInterface`

Points of attention.

1. `$handledStamp = $envelope->last(HandledStamp::class);`
2. `RegisterBetHandler` just implements `MessageHandlerInterface`
3. `BetResultHandler` handles an interface (`GameResultMessage`), on a specific bus (`event_bus`) and will call a different method (`handleResult`).

Exercise: Let's add a "Delete" button on the list of bets.

Middleware(s)

Middleware

```php
namespace App\Middleware;

use Symfony\Component\Messenger\Middleware\MiddlewareInterface;

class AuditMiddleware implements MiddlewareInterface
{
    public function handle(Envelope $message, StackInterface $stack): Envelope
    {
        try {
            echo sprintf('Started with message "%s"'."\n", get_class($envelope->getMessage()));

            return $stack->next()->handle($envelope, $stack);
        } finally {
            echo sprintf('Ended with message "%s"'."\n", get_class($envelope->getMessage()));
        }
    }
}
```

MessageBus == some middleware.

Middleware ¶

What happens when you dispatch a message to a message bus depends on its collection of middleware (and their order). By default, the middleware configured for each bus looks like this:

1. `add_bus_name_stamp_middleware` - adds a stamp to record which bus this message was dispatched into;
2. `dispatch_after_current_bus` - see Transactional Messages: Handle New Messages After Handling is Done;
3. `failed_message_processing_middleware` - processes messages that are being retried via the failure transport to make them properly function as if they were being received from their original transport;
4. Your own collection of middleware;
5. `send_message` - if routing is configured for the transport, this sends messages to that transport and stops the middleware chain;
6. `handle_message` - calls the message handler(s) for the given message.

Built-in middleware you can use

Built-in middleware you can use

1. `validation`.
   Runs the validator on your message.

Built-in middleware you can use

1. `validation`.
   Runs the validator on your message.

2. `doctrine_transaction`,
   `doctrine_clear_entity_manager`, and a few other from DoctrineBundle.

Built-in middleware you can use

1. `validation`.
   Runs the validator on your message.

2. `doctrine_transaction`,
   `doctrine_clear_entity_manager`, and a few other from DoctrineBundle.

3. All the default ones!

Add your middleware

```yaml
framework:
    messenger:
        buses:
            my_bus:
                middleware:
                    - 'App\\Middleware\\AuditMiddleware'
```

Envelopes

Add extra "non-domain" information to your messages.

Envelopes

```php
use Symfony\Component\Messenger\Envelope;
use Symfony\Component\Messenger\Stamp\SerializerStamp;

$bus->dispatch(
    (new Envelope($message))->with(new SerializerStamp([
        'groups' => ['my_serialization_groups'],
    ]))
);
```

Envelopes

```
class MyOwnMiddleware implements MiddlewareInterface
{
    public function handle($envelope, StackInterface $stack)
    {
        $envelopeItem = $envelope->last(AnotherStamp::class);

        // ...

        return $stack->next()->handle(
            $message->with(new AnotherStamp(/* ... */)),
            $stack
        );
    }
}
```

Envelopes

```php
class MyOwnMiddleware implements MiddlewareInterface
{
    public function handle($envelope, StackInterface $stack)
    {
        $envelopeItem = $envelope->last(AnotherStamp::class);

        // ...

        return $stack->next()->handle(
            $message->with(new AnotherStamp(/* ... */)),
            $stack
        );
    }
}
```

Envelopes

```php
class MyOwnMiddleware implements MiddlewareInterface
{
    public function handle($envelope, StackInterface $stack)
    {
        $envelopeItem = $envelope->last(AnotherStamp::class);

        // ...

        return $stack->next()->handle(
            $message->with(new AnotherStamp(/* ... */)),
            $stack
        );
    }
}
```

Envelopes

```php
class MyOwnMiddleware implements MiddlewareInterface
{
    public function handle($envelope, StackInterface $stack)
    {
        $envelopeItem = $envelope->last(AnotherStamp::class);

        // ...

        return $stack->next()->handle(
            $message->with(new AnotherStamp(/* ... */)),
            $stack
        );
    }
}
```

Envelopes

```
class MyOwnMiddleware implements MiddlewareInterface
{
    public function handle($envelope, StackInterface $stack)
    {
        $envelopeItem = $envelope->last(AnotherStamp::class);

        // ...

        return $stack->next()->handle(
            $message->with(new AnotherStamp(/* ... */)),
            $stack
        );
    }
}
```

Exercise: Let's create our own "audit" middleware.

It will log a message when starting handling and when finishing handling.

Transports

Bring some asynchronicity to your messages.

Transports

Transports

1. **Built-in AMQP, Doctrine and Redis transports**. You don't need anything more than the component and the PHP extension.

Transports

1. **Built-in AMQP, Doctrine and Redis transports**. You don't need anything more than the component and the PHP extension.
2. **Look at Enqueue**. Supports many cloud-based transports and Kafka, Gearman, etc...
   https://github.com/sroze/messenger-enqueue-transport

Transports

1. **Built-in AMQP, Doctrine and Redis transports**. You don't need anything more than the component and the PHP extension.

2. **Look at Enqueue**. Supports many cloud-based transports and Kafka, Gearman, etc...
   https://github.com/sroze/messenger-enqueue-transport

3. **Create your own.**
   Fully extendable by registering "factories".


@samuelroze - #websc

RabbitMq 🐇

You have it in the VM, or...

```
docker run -d --hostname rabbit --name rabbit -p
 15672:15672 -p 5672:5672 rabbitmq:3-management
```

```
sudo apt-get install php-amqp
```

Exercise: Your `Won` and `Lost` messages should go through the queue.

Exercise: Log a different message if your message has been just dispatched, received, sent and handled.

Things will fail...

You can configure the retry strategy

```yaml
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            myqueue: '%env(MESSENGER_DSN)%'

            retry_strategy: # (this is the default)
                max_retries: 3
                delay: 1000 # ms
                multiplier: 2 # 1, 2, 4, 8, ...
```

Failure transport to catch issues

```yaml
# config/packages/messenger.yaml
framework:
    messenger:
        failure_transport: failed

        transports:
          failed: 'doctrine://default?queue_name=failed'
          # ...
```

Manage your failed messages

Manage your failed messages

- `$ bin/console messenger:failed:show`

Manage your failed messages

- `$ bin/console messenger:failed:show`
- `$ bin/console messenger:failed:retry`

Manage your failed messages

- `$ bin/console messenger:failed:show`
- `$ bin/console messenger:failed:retry`
- `$ bin/console messenger:failed:remove {id}`

Exercise: Make your handler randomly fail and manage the failures.

Any question?

Thank you!

https://github.com/sroze/messenger-workshop